Federating Consistency for Partition-Prone Networks

Benjamin Bengfort and Pete Keleher
Department of Computer Science
University of Maryland, College Park, MD, USA
{bengfort,keleher}@cs.umd.edu

I. Introduction

Groups of strongly consistent devices can efficiently order events under ideal (data center) conditions, but become less effective in dynamic and heterogeneous environments. Weakly consistent devices efficiently tolerate both faults and dynamic conditions but are slow to converge on a single ordering of system events.

We propose "federated consistency", which combines the strengths of both approaches into a single protocol. Federated groups use a strongly consistent inner core of devices to maintain a totally ordered, fault-tolerant sequence of events. A cloud of weakly-consistent devices disseminates orderings and enables progress despite varying connectivity and partitions. Though the constituent sub-protocols take different (nearly opposite) approaches to resolving conflicts; we show that expanding distributed version vectors with a *forte* component allows them to inter-operate effectively. We use a discrete event simulation to show that a group of federated devices can obtain the key advantages of both approaches. Such systems have been investigated before [1], [2], but our approach targets more active "weak nodes" in a wide-area setting.

A. Performance Issues, and a Solution

Straightforward integration of eventual [3] (weakly-consistent) and Raft [4] (strongly-consistent) nodes turn out to perform worse than a cloud of either in isolation. The problem is that eventual and Raft replicas resolve write conflicts in exactly opposite ways. Eventual replicas choose the last of a set of conflicting writes through a latest-writer-wins policy, whereas Raft replicas effectively choose the first by dropping any write that conflicts with previously seen writes.

Given conflicting writes w_i with timestamp t and w_j with timestamp $t+1$, eventual replicas will converge to w_j because its timestamp is later[1]. However, the Raft nodes will converge to whichever write first reaches the leader, and there is no mechanism by which to override a write that has already been committed. The end result is that eventual replicas might all converge on w_j and Raft replicas on w_i, with neither write ever being replicated across the entire system. This disconnect arises from a fundamental mismatch in the protocols' approaches to conflict resolution. We could modify one or the other, but might then have a protocol that performs less well in a non-federated environment. We resolve this issue by noting that if the strong central quorum can make a write accepted by the Raft replicas "more recent" than any conflicting write, all eventual replicas will converge to the write chosen by the Raft replicas.

We therefore extend version vectors with an additional monotonically increasing component called the *forte* (strong) number, which can only be incremented by the leader of the Raft quorum. Because the Raft leader drops forks, or any version that was not more recent than the latest commit version, incrementing the forte number on commit ensures that only consistent versions have their forte numbers incremented. Version comparisons are performed by comparing forte numbers first, and then the timestamp, allowing Raft to "bump" its chosen version to a later timestamp than any conflicting writes. This forte bump must be propagated to derived writes as well. Otherwise, the increment of a write's forte number would result in writes derived from the write being erroneously identified as conflicting. On receipt of a version with a higher forte than the local, eventual replicas search for the forte entry in their local log, find all children of the update, and set the child version's forte equal to that of the parent.

Eventual replicas will converge on writes with a higher forte number, incorporating information from the central quorum. When partitioned, the forte number does not prevent progress in either the Raft quorum or the eventual cloud, and when connectivity is restored all replicas will converge to a single sequence of events.

II. Simulation Results

A. System Model

We target dynamic, geo-replicated distributed system models that co-locate replicas with clients rather than traditional cloud-service oriented approaches. We posit a file system as the natural use case of local, client-oriented systems, though the model easily generalizes to any distributed storage system.

B. Experiments

We investigate the effect of variable latency and the network environment on consistency by constructing a fully connected topology of replicas distributed among several geographic regions. Within each region, replicas enjoy stable, low-latency connections with their neighbors. The latency is higher and the connections more variable across regions, meaning that

[1]The timestamps need not reflect real time, only local event ordering.

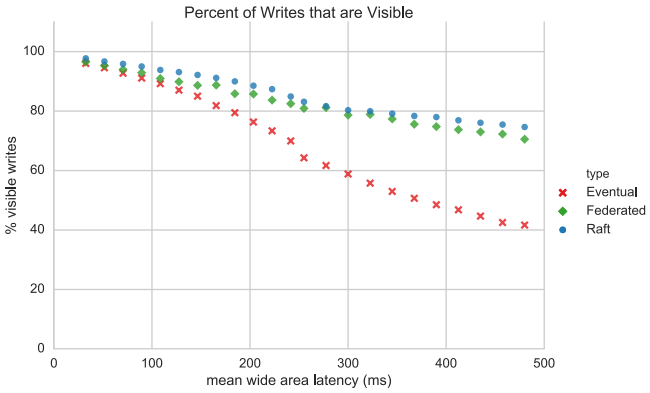

Fig. 1. The percentage of fully visible writes.

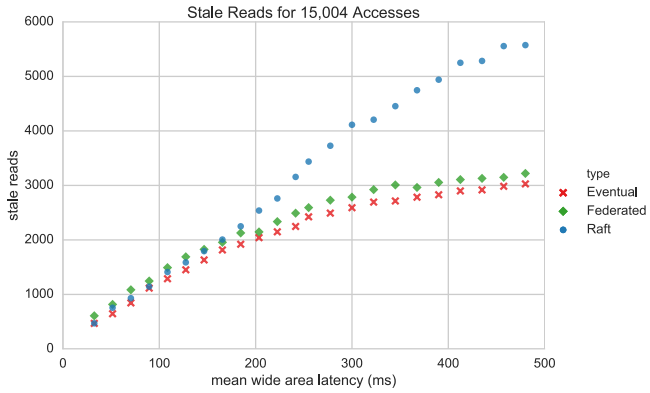

Fig. 2. The percent of reads that are stale in the system.

out of order messages are more common across the wide area than in the local area. We built a discrete event network simulation that allows us to flexibly configure network and workload parameters. Our eventual replicas replicate via anti-entropy with random neighbors selected for gossip. Raft nodes implement the Raft consensus protocol, using the replicated log to build a total ordering on committed writes.

C. Results and Metrics

In this paper we concentrate on two of our latency variation results. Each consistency protocol is parameterized by a timing variable T that is a function of the wide area latency mean (λ_μ) and standard deviation (λ_σ). The simulation workload was fixed with continuous accesses per replica at a rate that created the potential for access conflict. In effect, this meant that for approximately half the simulations (with the higher latencies), it was impossible for a write to become visible on another replica before a fork. Figure 1 shows that write propagation is much faster and more effective in Raft than in eventual, especially as network conditions deteriorate, and closely tracks the number of writes fully replicated by Raft.

The strong inner core of Raft replicas is the key to the federated protocol tracking Raft's performance. Eventual replicas are biased in favor of performing anti-entropy with local replicas, allowing most anti-entropy sessions to perform quickly and without delay. By contrast, the Raft replicas in our federated topology are intentionally spread across geographic regions. A new write originating at an eventual replica is quickly spread to the local Raft replica, and is then broadcast to the rest of the regions via the Raft `AppendEntries` message. Disseminating writes quickly minimizes the possibility of another, later eventual write starting up concurrently. Additionally, the *forte* number prevents new forked writes from stomping on a conflicting write disseminated via Raft replicas.

Figure 2 shows the average number of stale reads and forked writes across different mean latencies. All three protocols perform similarly at smaller latencies, but eventual and federated deal with high latencies much more effectively than Raft, at least for this size of system.

Higher latencies affect Raft in at least two ways. First, higher latency variability causes more out of order messages. Second, we parameterize system timeouts by T which, in turn, is based on mean latencies. The result is that Raft's `AppendEntries` delay is longer for simulations with higher mean latencies, resulting in more conflicts. The same is true for anti-entropy delays, but the speed of Raft decisions is determined by the slowest quorum member, which can be quite slow when message variability is large. By contrast, a slow anti-entropy participant only affects direct anti-entropy partners.

III. Discussion & Conclusion

This paper has presented a model for federating consistency, which allows individual replicas to expose local policies to users, while still allowing for global guarantees. We evaluate federating consistency in the context of a geographically dispersed wide-area file system. Our results show that a key to the global guarantees is in using a core strongly-consistent group to serialize and broadcast system writes. By designing a federated system where only the interactions between replicas of varying consistency types are defined, systems can scale beyond the handful of devices usually described to dozens or hundreds of replicas in variable-latency, partition-prone geographic networks. Replicas can monitor their local environment and adapt as necessary to meet timeliness and correctness constraints required by the local user.

References

[1] J. Gray, P. Helland, P. O'Neil, and D. Shasha, "The dangers of replication and a solution," in *ACM SIGMOD Record*, vol. 25. ACM, 1996, pp. 173–182.

[2] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, and others, "Oceanstore: An architecture for global-scale persistent storage," *ACM Sigplan Notices*, vol. 35, no. 11, pp. 190–201, 2000.

[3] D. B. Terry, A. J. Demers, K. Petersen, M. J. Spreitzer, M. M. Theimer, and B. B. Welch, "Session guarantees for weakly consistent replicated data," in *Parallel and Distributed Information Systems, 1994., Proceedings of the Third International Conference on*. IEEE, 1994, pp. 140–149.

[4] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, 2014, pp. 305–319.